統合開発環境 e² studio
e² studio での CUnit の使用方法（CC-RX）

はじめに
CUnit は C で単体テストを作成、管理、実行するためのシステムです。
このドキュメントでは、e² studio 環境で CUnit を使用した単体テストを実施する方法について説明します。

目次
1. 概要 ................................................................................................................................. 2
   1.1 目的 .......................................................................................................................... 2
   1.2 動作確認環境 ........................................................................................................... 2
2. セットアップ .................................................................................................................. 3
   2.1 CUnit をインストールする ...................................................................................... 3
   2.2 CUnit ライブラリを作成する .................................................................................. 3
3. 使用方法 ....................................................................................................................... 7
   3.1 テスト対象プロジェクトを作成する ........................................................................ 7
   3.2 CUnit を使用した単体テストを実行する ................................................................. 12
4. 参考情報 ........................................................................................................................ 14
   4.1 Web サイト .............................................................................................................. 14
   4.2 他のデバイス、コンパイラ、またはデバッグ環境を使用する場合 .......................... 14
改訂記録 ............................................................................................................................ 15
1. 概要

1.1 目的

e² studio は、Eclipse をベースとした統合開発環境です。様々なオープンソースソフトウェアのプラグインを組み込んで、機能を追加／拡張することができます。

CUnit は、C で単体テストを記述、管理、および実行するためのシステムです。ユーザーのテストコードにリンクするための静的ライブラリとして構築します。テスト構造を構築するためのシンプルなフレームワークを使用し、一般的なデータ型をテストするための豊富なアサーションセットを提供します。さらに、テスト実行結果のレポート作成のためのインターフェースも提供しています。

このドキュメントでは、e² studio 上で CUnit を使用した単体テストの実施方法について説明します。

1.2 動作確認環境

このドキュメントで説明する操作手順については、弊社にて以下の環境で確認を実施しています。ただし、オープンソースのソフトウェアとの連携になりますので、弊社が動作を保証するものではありません。あらかじめご了承の程お願い申し上げます。

[OS]
- OS　　Windows10（日本語版）

[ツール]
- e² studio　　2022-04
- CUnit　　2.1.2

[プロジェクト]
例として以下のデバイス、およびツールチェーンを指定したプロジェクトを使用します。
- デバイス　　RX610
- ツールチェーン　　CC-RX V3.04
2. セットアップ

この章では、CUnitをインストールしてセットアップする手順について説明します。

【注意事項】
- CUnit-2.1-2をダウンロードして使用してください。最新版であるCUnit-2.1-3にはいくつかの問題がありビルドエラーが発生する場合があります。
- コンパイラ（およびWindowsシステム）は、「curse」モジュールをサポートしません。

2.1 CUnitをインストールする

CUnitを「https://sourceforge.net/projects/cunit/files/CUnit/」からダウンロードします。圧縮ファイルを解凍して、任意のフォルダにCUnitパッケージを展開します。

2.2 CUnitライブラリを作成する

CUnitは、ユーザーのコードにリンクする静的ライブラリとして構築する必要があります。CUnitのライブラリプロジェクトを作成して、ライブラリを生成します。

1) メニュー［ファイル(F)］＞［新規(N)］＞［Renesas C/C++ Project］＞［Renesas RX］を選択します。
2) ［New C/C++ Project - Templates for New C/C++ Project］ダイアログが表示されます。右側リストボックスで「Renesas CC-RX C/C++ Library Project」を選択して、［次へ(N)］ボタンをクリックします。
3) ［New Renesas CC-RX Library Project - New Renesas CC-RX Library Project］ダイアログが表示されます。プロジェクト名編集ボックスに「CUnit」を入力して、［次へ(N)］ボタンをクリックします。
4) ［New Renesas CC-RX Library Project - Select toolchain, device & debug settings］ダイアログが表示されます。下記情報を設定します。残りの箇所は、デフォルトのままとします。設定を終了したら［終了(F)］ボタンをクリックします。
   - ［ツールチェーン:］コンボボックス：Renesas CCRX
   - ［ツールチェーン・バージョン:］コンボボックス：任意（このドキュメントでは「v3.04.00」を選択）
   - ［ターゲット・デバイス］編集ボックス：任意（このドキュメントでは「RX600 > RX610 > RX610 - 144pin > R5F56107VxFP」を選択）
5) Library プロジェクトが作成されて、[プロジェクト・エクスプローラー] ビューに、CUnit プロジェクトが表示されます。デフォルトのソースファイルは必要ないため、「src」フォルダ内のファイル (sample1.c、sample2.c、sample3.S) を削除します。

6) CUnit パッケージをインストールしたフォルダの「CUnit」フォルダにある、「¥Headers」、「¥Sources¥Basic」および「¥Sources¥Freamwork」を CUnit プロジェクトの「src」フォルダへコピーします。

プロジェクトは、次のようになります。

7) [プロジェクト・エクスプローラー] ビューで CUnit プロジェクトを選択して、コンテキスト・メニュー [C/C++ Project Settings Ctrl+Alt+P] を選択します。

8) [プロジェクト・エクスプローラー] ビューに CUnit プロジェクトを追加する場合、[ツール設定] タブの「ソース」を選択します。[ツール設定] タブの「ソース」を選び、「インクルード・ファイルを検索するフォルダ (-include)」リストボックスに「${workspace_loc:/${ProjName}/src/Headers}」を追加します。
図 4

9) [ツール設定] タブのツリーで「Compiler > ソース > 拡張」を選択します。 [C ソース (-lang)] コンポポックスで「C99 言語」を選択します。 [適用して閉じる] ボタンをクリックします。

図 5

10) 「Headers」フォルダに下記内容のファイルを作成して追加します。
（CC-RX は、time.h をサポートしていませんのでダミーのファイルを作成する必要があります）

- time.h

```c
#ifndef TIME_H
#define TIME_H_

typedef int clock_t;
#define CLOCKS_PER_SEC 1000
#define clock() (0)

#endif  /* TIME_H */
```
11) [プロジェクト・エクスプローラー] ビューで CUnit プロジェクトを選択して、コンテキスト・メニュー [プロジェクトのビルド(B)] を選択します。ビルドが実行され、「Debug」フォルダに「CUnit.lib」ファイルが表示されます。

図 6

CUnit ライブラリ・ファイル「CUnit.lib」は、CUnit テストフレームワークを提供します。同一コンパイラのプロジェクトであれば、どの C/C++プロジェクトでも使用することができます。
3. 使用方法

この章では、CUnitを使用したテスト実行のプロジェクトの作成方法、およびテストの実行方法について説明します。

3.1 テスト対象プロジェクトを作成する

テスト対象プロジェクトは、Renesas CC-RX C/C++ Executable Projectとして作成します。

1) メニュー［ファイル(F)］＞［新規(N)］＞［Renesas C/C++ Project］＞［Renesas RX］を選択します。
2) ［New C/C++ Project - Templates for Renesas RX Project］ダイアログが表示されます。右側リストボックスで「Renesas CC-RX C/C++ Executable Project」を選択します。［次へ(N)＞］ボタンをクリックします。

図 7

3) ［New Renesas CC-RX Executable Project - New Renesas CC-RX Executable Project］ダイアログが表示されます。［プロジェクト名(P):］編集ボックスに「SampleCUnit」を入力して、［次へ(N)＞］ボタンをクリックします。
4) ［New Renesas CC-RX Executable Project - Select toolchain, device & debug settings］ダイアログが表示されます。下記情報を設定します。残りの箇所は、デフォルトのままとします。設定を終了したら［次へ(N)＞］ボタンをクリックします。
  • ［ツールチェーン:］コンボボックス：Renesas CCRX
  • ［ツールチェーン・バージョン:］コンボボックス：任意（このドキュメントでは「v3.04.00」を選択）
  • ［ターゲット・デバイス］編集ボックス：任意（このドキュメントでは「R5F56107VxFP」を選択 [RX600 > RX610 > RX610 - 144pin > R5F56107VxFP] ）
  • ［Hardware Debug 構成を生成］チェックボックスのチェックをはずす。
  • ［Debug 構成を生成］チェックボックスをチェックし、「RX Simulator」を選択する。
5) [New Renesas CC-RX Executable Project - コーディング・アシストツールの選択] ダイアログが表示されますので、[次へ(N) >] ボタンをクリックします。

6) [New Renesas CC-RX Executable Project - Settings The Contents of Files to be Generated] ダイアログが表示されます。[Renesas デバッグ仮想コンソールを使用する] チェックボックスをチェックして、[終了(F)] ボタンをクリックします。

7) 「src」フォルダに下記内容のファイルを作成します

- source.h

```c
#ifndef SOURCE_H_
#define SOURCE_H_
#endif
```
int add(int a, int b);
int subtract(int a, int b);

/* SOURCE_H */

source.c
#include "source.h"

int add(int a, int b) {
    return a + b;
}

int subtract(int a, int b) {
    return a - b;
}

testsource.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include "CUnit.h"
#include "source.h"

// This is a test case used to test add() function in source.c
static void test_Add_01(void) {
    // Equal Assertion is used in this test case.
    // 1 is expected value, and add(1,0) is actual return value.
    // If expected value is not same, assertion occurs.
    // We can refer the Reference document for the other useful assertion.
    CU_ASSERT_EQUAL(1, add(1,0));
}

static void test_Add_02(void) {
    CU_ASSERT_EQUAL(10, add(1,9));
}

// This is a test case used to test subtract() function in source.c
static void test_Subtract(void) {
    // 0 is expected value, and subtract(1,1) is actual return value.
    // If expected value is not same, assertion occurs.
    CU_ASSERT_EQUAL(0, subtract(1,1));
}

// This is a test suite
static CU_TestInfo tests_Add[] = {
    // Register test case to test suite
    {"test_Add_01", test_Add_01},
    {"test_Add_02", test_Add_02},
    CU_TEST_INFO_NULL,
};
統合開発環境 e² studio での CUnit の使用方法 (CC-RX)

8) 既存ファイル「SampleCUnit.c」の内容を、テストを実行するための下記コードに置き換えます。

- SampleCUnit.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Basic.h"

void exit(long);
void abort(void);
int main(void);
extern void AddTests();

int main(void) {
    // Define the run mode for the basic interface
    // Verbose mode - maximum output of run details
    CU_BasicRunMode mode = CU_BRM_VERBOSE;

    // Define error action
    // Runs should be continued when an error condition occurs (if possible)
    CU_ErrorAction error_action = CUEA_IGNORE;

    // Initialize the framework test registry
    if (CU_initialize_registry()) {
        printf("Initialization of Test Registry failed.\n");
    } else {
        // Call add test function
```
AddTests();

   // Set the basic run mode, which controls the output during test runs
   CU_basic_set_mode(mode);

   // Set the error action
   CU_set_error_action(error_action);

   // Run all tests in all registered suites
   printf("Tests completed with return value \%d.\n",
           CU_basic_run_tests());

   // Clean up and release memory used by the framework
   CU_cleanup_registry();

   return 0;
}

void abort(void) {}
void exit(long exitcode) {}

9) 既存ファイル「sbrk.h」の内容を、テストを実行するための下記コードに置き換えます。

・ sbrk.h

/* size of area managed by sbrk */
#define HEAPSIZE 0x800

10) [プロジェクト・エクスプローラー] ビューで SampleCUnit プロジェクトを選択して、コンテキスト・メニュー [C/C++ Project Settings Ctrl+Alt+P] を選択します。

11) [プロパティ: SampleCUnit] ダイアログが表示されます。[ツール設定] タブのツリーで「Compiler > ソース」を選択します。[インクルード・ファイルを検索するフォルダ (-include)] リストボックスに「${workspace_loc:/CUnit/src/Headers}」を追加します。

図 10
12) ［ツール設定］タブのツリーで「Linker > 入力」を選択します。［リンクするリロケータブル・ファイル、ライブラリ・ファイルおよびバイナリ・ファイル（-input/-library/-binary）］リストボックスに
「${workspace_loc:/CUnit/Debug/CUnit.lib}」を追加します。

図 11

13) ［ツール設定］タブのツリーで「Library GeneratorLinker > 構成」を選択します。［C 言語標準ライブラリ関数の構成（-lang）］コンボボックスで「C89 規格および C99 規格準拠」を選択します。次に、
[ctype.h (C89/C99): 文字操作用ライブラリ（-head=type）]チェックボックスをチェックします。そして、［適用して閉じる］ボタンをクリックします。

図 12

14) ［プロジェクト・エクスプローラー］ビューで SampleCUnit プロジェクトを選択して、コンテキスト・メニュー［プロジェクトのビルド (B)］を選択します。ビルドが実行され、「バイナリ」フォルダに
「SampleCUnit.abs」ファイルと「SampleCUnit.x」ファイルが表示されます。

3.2 CUnit を使用した単体テストを実行する
1) メニュー［実行 (R)］＞［デバッグの構成 (B)］を選択します。
2) ［デバッグ構成］ダイアログが表示されます。「Renesas Simulator Debugging > SampleCUnit Debug」を選択して、［デバッグ (D)］ボタンをクリックします。
3) シミュレータが起動します。メニュー ［Renesas View］＞［デバッグ］＞［Renesas Debug Virtual Console］を選択します。
4) [Renesas Debug Virtual Console] ビューが表示されます。メニュー [実行(R)] > [再開(M)] を何度か選択すると、「void Excep_BRK(void) { wait(); }」でプログラムが停止します。

5) 以下のように、[Renesas Debug Virtual Console] ビューにテスト結果が表示されます。

![テスト結果](image)

図 13 [ターミナル] ビューのテスト結果表示
4. 参考情報

4.1 Web サイト

- e² studio
  https://www.renesas.com/software-tool/e-studio
- CUnit
  http://cunit.sourceforge.net/

4.2 他のデバイス、コンパイラ、またはデバッグ環境を使用する場合

このドキュメントは、CC-RX 用シミュレーション環境と printf を組み合わせた環境を前提とした説明になっていますが、Arm 系コア向けのデバッガでは semi-hosting 機能等によりコンソール出力が可能です。また、エミュレータがコンソール出力機能を持たず printf での出力が行えなくても「Dynamic printf」を利用すればコンソールへの表示は可能です。
「Dynamic printf」の使用方法は、下記ページのビデオをご確認いただけます。

「Dynamic printf」の使用方法
Renesas

[例]
以下のような自作の printf を作成してそこに「Dynamic printf」を指定すると、本ドキュメントと同様の結果を得ることができます。

- xprintf.h

```c
#ifndef XPRINTF_H_
#define XPRINTF_H_

#define printf xPrintf
void xPrintf(const char* format, ...);
#endif
```

- xprintf.c

```c
void xPrintf(const char* format, ...);

void xPrintf(const char* format, ...)
{
  static char szBuf[512];
  va_list ap;
  va_start(ap, format);
  vsprintf(szBuf, format, ap);
  va_end(ap);  /* here place Dynamic Printf as "%s",szBuf */
}
```
### 改訂記録

<table>
<thead>
<tr>
<th>Rev.</th>
<th>発行日</th>
<th>ページ</th>
<th>改訂内容</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.02</td>
<td>Jul.12.22</td>
<td>2ページ</td>
<td>動作環境を e2 studio 2022-04 に更新</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14ページ</td>
<td>「Dynamic printf」の説明を追加</td>
</tr>
</tbody>
</table>

"Dynamic printf" の説明を追加した。

動作環境を e2 studio 2022-04 に更新した。
製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS製品の取り扱いの際は静電気防止を心がけてください。CMOS製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーヤマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS製品を実装したボードについても同様の扱いをしてください。

2. 電源投入時の処置
   電源投入時の製品の状態は不定です。電源投入時には、LSIの内部回路の状態は不確定であり、レジスタの設定や各種のモードは不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまで、リセット端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。

3. 電源オフ時における入力信号
   異常電流電流入力端子を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」について記載のある製品は、その内容を守ってください。

4. 未使用端子の処理
   未使用端子は、未使用端子の処理に従って処理してください。CMOS製品の入出力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI周辺のノイズが増加され、LSI内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。

5. クロックについて
   リセット時は、クロックが安定した後に、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックを開始するシステムでは、クロックが十分安定した後に、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え前のクロックが十分安定してから切り替えてください。

6. 入力信号の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS製品の入力ノイズなどに起因して、VIL（Max.）からVOS（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが安定の場合はもちろん、VIL（Max.）からVOS（Min.）までの領域を通過する遅延時間中にチャタリングノイズなどが入らないように使用してください。

7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。

8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパーソンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。


ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報使用の際、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者が生じた損害を含みます。以下同じです。）に関し、当社は、一切その責任を負いません。

2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路図等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に関する権利の侵害について、当社は、一切その責任を負いません。

3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を侵害する行為を行った場合、当該ライセンス取得の判断および取引はお客様の責任において行ってください。

4. 当社製品を、改変、複製、リバースエンジニアリング、その他不適切な使用を行った場合、かかる改変、複製、リバースエンジニアリング等により損害が発生した場合、当社は一切その責任を負いません。

5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

6. あらゆる半導体製品は、外部攻撃からの安全性を100%保証されているわけではありません。当社ハードウェア/ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによっても、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されるあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵害行為（「脆弱性問題」といいます。）によって影響を受けないことを保証いたしません。

7. あらゆる半導体製品は、外部攻撃からの安全性を100%保証されているわけではないので、当社ハードウェア/ソフトウェア製品はセキュリティ対策が組み込まれているものもありますが、これによっても、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されるあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵害行為（「脆弱性問題」といいます。）によって影響を受けないことを保証いたしません。

8. あらゆる半導体製品は、外部攻撃からの安全性を100%保証されているわけではないので、当社ハードウェア/ソフトウェア製品はセキュリティ対策が組み込まれているものもありますが、これによっても、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されるあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵害行為（「脆弱性問題」といいます。）によって影響を受けないことを保証いたしません。

9. あらゆる半導体製品は、外部攻撃からの安全性を100%保証されているわけではないので、当社ハードウェア/ソフトウェア製品はセキュリティ対策が組み込まれているものもありますが、これによっても、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されるあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵害行為（「脆弱性問題」といいます。）によって影響を受けないことを保証いたしません。

10. あらゆる半導体製品は、外部攻撃からの安全性を100%保証されているわけではないので、当社ハードウェア/ソフトウェア製品はセキュリティ対策が組み込まれているものもありますが、これによっても、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されるあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵害行為（「脆弱性問題」といいます。）によって影響を受けないことを保証いたしません。

11. あらゆる半導体製品は、外部攻撃からの安全性を100%保証されているわけではないので、当社ハードウェア/ソフトウェア製品はセキュリティ対策が組み込まれているものもありますが、これによっても、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されるあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵害行為（「脆弱性問題」といいます。）によって影響を受けないことを保証いたしません。

12. あらゆる半導体製品は、外部攻撃からの安全性を100%保証されているわけではないので、当社ハードウェア/ソフトウェア製品はセキュリティ対策が組み込まれているものもありますが、これによっても、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されるあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵害行為（「脆弱性問題」といいます。）によって影響を受けないことを保証いたしません。

13. あらゆる半導体製品は、外部攻撃からの安全性を100%保証されているわけではないので、当社ハードウェア/ソフトウェア製品はセキュリティ対策が組み込まれているものもありますが、これによっても、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されるあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵害行為（「脆弱性問題」といいます。）によって影響を受けないことを保証いたしません。